

PR #53016 完整报告

vllm-project/vllm

[Bugfix] Skip MM processor cache inserts larger than capacity

合并时间: 2026-08-20 21:42

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/53016>

执行摘要

- 一句话: 跳过超限多模态缓存插入, 修复引擎启动崩溃
- 推荐动作: 建议精读。它不止是一个小 bugfix: review 中围绕 "方法归属 (基类选择)" 与 "API 边界 (LRUCache 是否该提供 put_if_fits)" 的两次迭代体现了 vLLM 对公共抽象位置 和性能敏感路径 (避免重复 getsizeof) 的考量, 值得在同类缓存 / 防崩溃修复中复用。合并前已由多模态领域维护者 DarkLight1337 批准, 测试覆盖了三类 LRU 缓存与 LRUCache API 本身, 可作为 "缓存容量保护" 修复的参考模板。

功能与动机

issue #52835 指出根因: `cachetools.Cache.__setitem__` 在单个条目超过缓存总容量 `maxsize` 时抛出 `ValueError("value too large")`, 而多模态处理器缓存 (`vllm/multimodal/cache.py`) 插入前从不检查大小; 最坏场景是 memory profiling 的 dummy 项在构造上就是最大尺寸条目, 一旦超过 `--mm-processor-cache-gb` 引擎连启动都做不到 (Qwen 长视频理解配方实测触发, video pixel budget 469,762,048, 单个处理项达数 GiB)。PR body 补充: SHM 缓存早已有超限跳过逻辑而 LRU 没有, 且超限条目永远不可能命中缓存, "evicting the entire cache to fail the insert is lose/lose"。

实现拆解

1. 底层 API (`vllm/utils/cache.py`): 为 LRUCache 新增 `put_if_fits(key, value) -> bool`。实现用 `try/except` 包裹 `self[key] = value`, 仅在异常消息精确匹配 "value too large" 时返回 `False`, 其余 `ValueError` 照常上抛; 成功路径只走一次 `__setitem__`, 大小由 `cachetools` 内部仅计算一次, 满足 review 中 "避免重复调用 `getsizeof`" 的要求, 且 `put()` 保持原抛错语义不变。
2. 统一入口 (`vllm/multimodal/cache.py`): 在 P0/P1 共用基类 `BaseMultiModalCache` 上 新增实例方法 `cache_if_fits`, 委托 `cache.put_if_fits`; 插入被跳过时用 `logger.warning_once` 输出一次告警, 内容包含条目大小 (`format_gib(cache.getsizeof(value))`) 和当前 `--mm-processor-cache-gb` 配置, 并给出调大配置的建议。放在该基类而非更窄的 `BaseMultiModalProcessorCache`, 是因为 P1 的 `MultiModalReceiverCache` 继承自 `BaseMultiModalReceiverCache`, 而 SHM 子类没有 `LRU_cache` (其 `put()` 已自带超限跳过)。
3. 三处插入点替换: `MultiModalProcessorOnlyCache.get_and_update_item` (存 `MultiModalProcessorCacheItem`)、`MultiModalProcessorSenderCache.get_and_updat`

e_item (存 metadata 的影子缓存)、MultiModalReceiverCache.get_and_update_item (P1 存原始 item) 全部从 self._cache[mm_hash] = ... 改为 self.cache_if_fits(...), 保证 P0/P1 镜像行为一致。

4. 测试与配套: tests/multimodal/test_cache.py 新增

test_oversized_item_is_served_uncached, 用 1 KiB 缓存 + 4 KiB 条目分别验证三类 LRU 缓存跳过超限项、小项仍可缓存; tests/utils_/test_cache.py 新增

test_lru_cache_put_if_fits, 锁定 put_if_fits 返回 False、不写入、且 put() 仍抛错的契约; vllm/config/multimodal.py 的配置 docstring 与 docs/configuration/optimization.md 同步说明新行为。

关键文件:

- vllm/multimodal/cache.py (模块 多模态缓存; 类别 source; 类型 core-logic; 符号 cache_if_fits): 核心修复所在: 在 BaseMultiModalCache 上新增 cache_if_fits 实例方法, 并将 processor-only、sender、receiver 三处 LRU 插入点统一替换为条件插入, 超限条目跳过缓存并原样返回。
- vllm/utils/cache.py (模块 缓存工具; 类别 source; 类型 core-logic; 符号 put_if_fits): 为 LRU 缓存新增 put_if_fits(key, value) -> bool 通用 API, 捕获 cachetools 的 "value too large" 异常并返回 False, 是修复的底层支撑。
- tests/multimodal/test_cache.py (模块 缓存测试; 类别 test; 类型 test-coverage; 符号 test_oversized_item_is_served_uncached): 新增 test_oversized_item_is_served_uncached, 直接覆盖 processor-only、sender、receiver 三类 LRU 缓存的超限跳过路径, 是本次回归的核心防线。
- tests/utils_/test_cache.py (模块 缓存测试; 类别 test; 类型 test-coverage; 符号 test_lru_cache_put_if_fits): 新增 test_lru_cache_put_if_fits, 锁定 put_if_fits 的返回契约及 put() 仍抛错的语义, 防止未来重构破坏行为。
- vllm/config/multimodal.py (模块 配置; 类别 source; 类型 configuration): 更新 mm_processor_cache_gb 配置 docstring, 说明超限条目将以未缓存方式服务并告警, 让用户从配置源头理解新行为。
- docs/configuration/optimization.md (模块 文档; 类别 docs; 类型 documentation): 同步优化指南文档, 说明超限条目不再导致启动失败, 并给出 mm_processor_cache_gb 的调优建议。

关键符号: cache_if_fits, put_if_fits, get_and_update_item

关键源码片段

vllm/multimodal/cache.py

核心修复所在: 在 BaseMultiModalCache 上新增 cache_if_fits 实例方法, 并将 processor-only、sender、receiver 三处 LRU 插入点统一替换为条件插入, 超限条目跳过缓存并原样返回。

vllm/multimodal/cache.py —— BaseMultiModalCache 新增的 cache_if_fits 与调用点

```
def cache_if_fits(
```

```

self,
cache: LRUCache[str, _V],
key: str,
value: _V,
) -> bool:
    """Insert `value` if it fits in `cache`.

    cachetools 在单个条目超过 maxsize 时抛 ValueError("value too large"),
    即使缓存此刻是空的; 超限条目永远不可能命中, 因此跳过插入并
    按未缓存处理, 而不是让 EngineCore 启动崩溃。
    """
    if cache.put_if_fits(key, value):
        return True

    # 只有超限告警路径才需要再算一次大小, 正常插入只算一次
    logger.warning_once(
        "Skipping multi-modal processor cache insert for an item of "
        "%s GiB because it exceeds --mm-processor-cache-gb=%s. "
        "The item will be processed uncached; increase "
        "--mm-processor-cache-gb to cache items of this size.",
        format_gib(int(cache.getsizeof(value))),
        format_gib(int(cache.maxsize)),
    )
    return False

# MultiModalProcessorOnlyCache / SenderCache / ReceiverCache 三处插入点
# 统一由 self._cache[mm_hash] = ... 改为条件插入, 保证 P0/P1 镜像一致
@override
def get_and_update_item(
    self,
    mm_item: MultiModalProcessorCacheInItem,
    mm_hash: str,
) -> MultiModalProcessorCacheOutItem:
    if (cached_item := self._cache.get(mm_hash)) is not None:
        return cached_item.item, cached_item.prompt_updates

    assert mm_item is not None, f"Expected a cached item for {mm_hash=}"

    # 超限时跳过插入并原样返回 item, 避免缓存写入抛异常
    self.cache_if_fits(
        self._cache, mm_hash, MultiModalProcessorCacheItem(*mm_item)
    )
    return mm_item

```

vllm/utils/cache.py

为 LRUCache 新增 put_if_fits(key, value) -> bool 通用 API, 捕获 cachetools 的 "value too large" 异常并返回 False, 是修复的底层支撑。

vllm/utils/cache.py —— LRUCache 新增的 put_if_fits

```
def put_if_fits(self, key: _K, value: _V) -> bool:
    """Insert `value` if it is not larger than the cache capacity.

    Unlike `put`, this does not raise when a single item exceeds
    `maxsize`. Size is computed once inside the insert path.
    """
    try:
        self[key] = value
    except ValueError as e:
        # cachetools 仅在条目超过 maxsize 时抛出 ValueError("value too large"),
        # 其余 ValueError 需继续上抛，避免吞掉真实错误
        if str(e) != "value too large":
            raise
        return False
    return True
```

评论区精华

DarkLight1337 对最初的 classmethod 方案提出两点设计意见：一是希望把超限跳过逻辑做成 `BaseMultiModalProcessorCache` 的实例方法；作者解释 P1 的 `MultiModalReceiverCache` 继承的是 `BaseMultiModalReceiverCache`，SHM 子类又没有 `LRU_cache`，放在该 ABC 上会要么用不上、要么被迫伪造缓存，最终按建议放到二者共同基类 `BaseMultiModalCache`。二是建议由 `LRUCache` 自身提供返回布尔值的插入方法，避免重复调用 `getsizeof`；作者据此新增 `LRUCache.put_if_fits`，只在超限告警路径才再次计算大小。另有 docstring 从 Sphinx 双反引号改为 Markdown 代码跨度的风格修正。最终 DarkLight1337 批准："Thanks for your patience, LGTM now"。

- `cache_if_fits` 的归属位置（classmethod vs 基类实例方法）（design）：DarkLight1337 建议放到两者共同基类 `BaseMultiModalCache`，作者照做并确认 SHM 子类不调用该方法
- `LRUCache` 层面提供返回布尔的插入方法，避免重复 `getsizeof`（performance）：新增 `LRUCache.put_if_fits -> bool`，`cache_if_fits` 委托该 API，性能与语义均满足要求
- docstring 风格（markdown 而非 sphinx）（style）：作者改为 Markdown code spans，提交信息确认修改完成

风险与影响

- 风险：
 1. 异常消息字符串匹配的脆弱性：`put_if_fits` 依赖 `str(e) != "value too large"` 识别超限异常，对 `cachetools` 实现细节有隐式耦合；若上游改动消息文本，可能误吞其他 `ValueError` 或回退为崩溃。现有单测锁定了这条消息，后续升级 `cachetools` 时需关注。
 2. P0/P1 镜像一致性隐患：P0 sender 缓存存的是 `MultiModalProcessorCacheItemMeta` data，P1 receiver 存原始 `mm_item`，两者 `getsizeof` 结果不同，可能出现 P0 判定放得下、P1 判定放不下的不对称；该场景会走 `MultiModalCacheMissError` 恢复路径（`test_mm_cache_miss_raises_and_recovers` 已覆盖），表现为一次额外 IPC 重发而非

致命错误，但值得留意。

3. 行为语义变化：超限条目现在每次请求都会重新执行处理器且不写缓存，对超大视频 / 图像场景有重复预处理开销；这是 " 崩溃 vs. 性能 " 之间的正确取舍，告警信息明确指向调大 `--mm-processor-cache-gb`。 - 影响：影响范围：修复了长视频 / 高分辨率多模态场景下 EngineCore 无法启动的问题（如 Qwen 长视频理解配方配默认 `mm_processor_cache_gb=4`）；对原本正常工作的用户，缓存命中 / 未命中语义完全不变。对系统而言，超限项每请求重新处理会带来重复预处理开销，但避免了启动硬失败；告警信息直接指导用户调大配置。对团队而言，LRUCache 新增的 `put_if_fits` 是通用 API，未来同类 " 放不下就跳过 " 的场景可复用；`vllm/multimodal/cache.py` 中 P0/P1 与 SHM 两条缓存路径的差异已在注释中明确说明，降低后续维护困惑。 - 风险标记：启动路径修复，依赖 `cachetools` 异常消息，P0/P1 镜像一致性，超限项不再缓存

关联脉络

- PR #51703 [Bugfix] Record non-ImportError attention backend probe failures instead of crashing engine init: 同为 " 引擎初始化 / 启动阶段的边缘条件从硬崩溃改为可记录或可恢复 " 的健壮性修复模式，与本 PR 的 " 超限条目跳过缓存而非启动崩溃 " 理念一致。
- PR #51827 [3/N] Harden Transformers modelling backend multi-modal path: 同属多模态处理管线的稳定性改造方向，本 PR 为其缓存层补上防崩溃保护，二者共同推进该链路的健壮性。
- PR #53064 [Refactor] Remove InputPreprocessor: 多模态输入处理链路的结构性重构（输入处理直连 `Renderer`），与本 PR 改动区域（`multimodal processing/cache`）相邻，反映同一管线处于持续演进期。